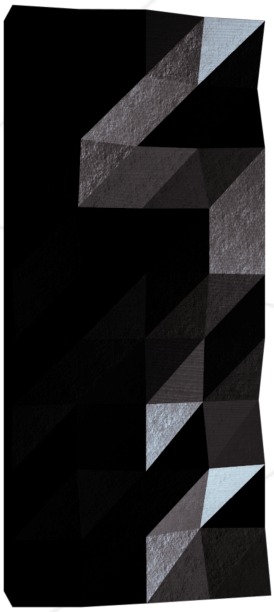

# get_next_line

## Parce que lire une ligne sur un fd, c'est chiant

Staff pedago pedago@42.fr

*Résumé: Ce projet a pour but de vous faire coder une fonction qui renvoit une ligne terminée par un retour à la ligne lue depuis un descripteur de fichier.*

# Table des matières


| | | |
|---|---|---|
| **I** | **Préambule** | **2** |
| **II** | **Introduction** | **3** |
| **III** | **Objectifs** | **4** |
| **IV** | **Consignes générales** | **5** |
| **V** | **Partie obligatoire** | **6** |
| **VI** | **Partie bonus** | **8** |
| **VII** | **Rendu et peer-évaluation** | **9** |

# Chapitre I

# Préambule

Suave, mari magno turbantibus aequora ventis,
e terra magnum alterius spectare laborem ;
non quia vexari quemquamst jucunda voluptas,
sed quibus ipse malis careas quia cernere suave est.
Suave etiam belli certamina magna tueri
per campos instructa, tua sine parte pericli.
Sed nihil dulcius est bene quam munita tenere
edita doctrina sapientum templa serena
despicere unde queas alios passimque videre
errare, atque viam palantis quaerere vitae,
certare ingenio, contendere nobilitate,
noctes atque dies niti praestante labore
ad summas emergere opes rerumque potiri.
Ô miseras hominum mentes, ô pectora caeca !
Qualibus in tenebris vitae quantisque periclis
degitur hoc aevi quodcumque est ! Nonne videre
nihil aliud sibi naturam latrare, nisi ut qui
corpore sejunctus dolor absit, mensque fruatur
jucundo sensu cura semota metuque ?

Un peu de culture générale, peuchère !

# Chapitre II

# Introduction

Vous commencez maintenant à vous rendre compte qu'à chaque fois que vous retrouvez dans la situation de lire des données depuis un file descriptor et que cette donnée n'est pas d'une taille connue à l'avance, c'est compliqué. Quelle taille de buffer choisir ? Combien de fois lire sur le file descriptor pour retrouver la donnée ?

Il est parfaitement naturel et commun en programmation de vouloir lire une "ligne" terminée par un retour à la ligne depuis un file descriptor. Par exemple chaque commande que vous tapez dans votre shell ou bien chaque ligne lue depuis un fichier plat.

Grâce au projet `get_next_line`, vous allez pouvoir écrire une bonne fois pour toute une fonction vous permettant de lire une ligne terminée par un retour à la ligne depuis un file descriptor, l'ajouter à votre `libft` si le coeur vous en dit, et surtout l'utiliser dans tous vos projets suivants qui en auront besoin.

# Chapitre III

# Objectifs

Ce projet va non seulement vous permettre d'ajouter une fonction très pratique à votre collection, mais vous permettra également d'aborder un nouvel élément surprenant de la programmation en `C` : les variables statiques.

Vous experimenterez également plus profondément les allocations, qu'elles aient lieu sur la pile ou sur le tas, la manipulation et le cycle de vie d'un buffer, et la complexité inattendue qu'implique l'utilisation d'une ou plusieurs variables statiques.

Bien entendu, vous renforcerez également votre rigueur grâce au respect de la Norme, mais aussi en découvrant que l'état initial d'une variable dans une fonction peut varier d'un appel à l'autre de cette fonction...

# Chapitre IV

# Consignes générales

- Vous ne devez rendre que deux fichiers : `get_next_line.c` et `get_next_line.h`
- Si vous êtes malin et que vous utilisez votre libft, rendez également votre dossier `libft` à la racine de votre rendu.
- Il ne doit y avoir aucune fonction `main` dans votre rendu.
- Ne rendez pas de `Makefile`.
- Votre projet doit être à la Norme.
- Vous devez gérer les erreurs de façon sensible. En aucun cas votre programme (ou dans le cas présent, votre fonction) ne doit quitter de façon inattendue (Segmentation fault, bus error, double free, etc).
- Toute mémoire allouée sur le tas doit être libérée proprement quand nécessaire.
- Votre projet ne doit pas avoir de fuites mémoire.
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
```

- Si vous choisissez de rendre ce projet en utilisant votre bibliothèque `libft`, il vous est formellement interdit d'y ajouter des fonctions spécifiques à votre rendu de `get_next_line` pour contourner les limitations de la Norme. celà sera considéré comme triche lors de la soutenance. Votre `get_next_line` doit tenir en 5 fonctions de 25 lignes maximum. Le respect de cette consigne sera méticuleusement vérifié en soutenance. Inutile de venir à la pédago demander si telle ou telle fonction est acceptable car vous voulez l'ajouter à votre bibliothèque. Demandez-vous plutot si votre fonction brise cette consigne ou non et servez-vous de cette chose disgracieuse située au dessus de vos épaules. Si vous respectez cette règle vous êtes bien entendu encouragés à étendre votre bibliothèque avec des fonctions génériques dont vous aurez découvert l'utilité au cours de ce projet.
- Les fonctions de la `libc` autorisées sur ce projet sont `read`, `malloc` et `free`.

# Chapitre V

# Partie obligatoire

- Ecrivez une fonction qui retourne une ligne lue depuis un file descriptor.
- On appelle “ligne” une suite de caractères terminée par un ’\n’ (code ascii 0x0a) ou bien par End Of File (EOF).

https://latedev.wordpress.com/2012/12/04/all-about-eof/ : cet article peut se montrer pertinent pour qui sait lire.

- Votre fonction aura le prototype suivant :

```
int     get_next_line(const int fd, char **line);
```

- Le premier paramètre est le file descriptor depuis lequel lire.
- Le second paramètre est l’adresse d’un pointeur sur caractère qui servira à stocker la ligne lue sur le file descriptor.
- La valeur de retour peut être 1, 0 ou -1 selon qu’une ligne a été lue, que la lecture est terminée ou bien qu’une erreur est survenue respectivement.
- Votre fonction get_next_line doit renvoyer son resultat sans le ’\n’.
- Un appel en boucle à votre fonction get_next_line permettra donc de lire le texte disponible sur un descripteur de fichier une ligne à la fois jusqu’à la fin du texte, quelque soit la taille du texte en question ou d’une de ses lignes.
- Assurez-vous que votre fonction se comporte bien lorsqu’elle lit depuis un fichier, depuis l’entrée standard, depuis une redirection, etc.
- Votre fichier get_next_line.h doit au moins contenir le prototype de la fonction get_next_line et une macro permettant de choisir la taille du buffer de lecture de read. Cette valeur sera modifiée en soutenance pour évaluer la robustesse de votre rendu. Cette macro devra impérativement s’appeler BUFF_SIZE. Par exemple :

```
#define BUFF_SIZE 32
```

Est ce que votre code fonctionne toujours si BUFF_SIZE vaut 9999 ? Et si BUFF_SIZE vaut 1 ? Et 10000000 ? Savez-vous pourquoi ?

- On considère que `get_next_line` a un comportement indeterminé si entre deux appels, un même descripteur de fichier désigne deux fichiers différents alors que la lecture du premier fichier n'était pas terminée.
- On considère également qu'un appel à la fonction `lseek(2)` n'aura jamais lieu entre deux appels à `get_next_line` sur un même descripteur de fichier.
- On considère enfin que `get_next_line` a un comportement indeterminé en cas de lecture dans un fichier binaire. Cependant, si vous le souhaitez, vous pouvez rendre ce comportement cohérent.
- Les variables globales sont interdites.
- Les variables statiques sont autorisées.

Savoir ce qu'est une variable statique est un bon début : https://en.wikipedia.org/wiki/Static_variable

# Chapitre VI

# Partie bonus

Le projet `get_next_line` est simple et laisse peu de latitudes pour ajouter des bonus, mais je suis certain que vous avez beaucoup d'imagination. Si vous avez réussi entièrement et parfaitement la partie obligatoire, cette section propose quelques pistes pour aller plus loin. Je repète, aucun bonus ne sera comptabilisé si la partie obligatoire n'est pas parfaite.

- Réussir `get_next_line` avec une seule variable statique.
- Pouvoir gérer plusieurs descripteurs de fichiers avec votre `get_next_line`. Par exemple, si les descripteurs de fichier 3, 4 et 5 sont accessibles en lecture, alors on peut appeler `get_next_line` une fois sur 3, une fois sur 4, à nouveau une fois sur 3, puis une fois sur 5, etc, sans perdre le fil de la lecture sur chacun des descripteurs.

# Chapitre VII

# Rendu et peer-évaluation

Rendez-votre travail sur votre dépot `GiT` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance et par la moulinette.

Une moulinette passera sur votre dépôt une fois vos soutenances effectuées. Elle compilera de cette manière :

```
$> make -C libft/ fclean && make -C libft/
```

Puis :

```
$> clang -Wall -Wextra -Werror -I libft/includes -o get_next_line.o -c get_next_line.c
$> clang  -Wall -Wextra -Werror -I libft/includes -o main.o -c main.c
$> clang -o test_gnl main.o get_next_line.o -I libft/includes -L libft/ -lft
```

Veillez à ce que votre dépôt compile bien de la même manière (le `main` est bien entendu le notre).

Bon courage à tous !